**附件：**

山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000130138 | 姓名：宋璎航 | | 班级： 20.3 |
| 实验题目：栈 | | | |
| 实验学时：2 | | 实验日期：2021/11/4 | |
| 实验目的：   1. 掌握栈结构的定义与实现； 2. 掌握栈结构的使用。 | | | |
| 软件开发环境：  VSC | | | |
| 1. 实验内容   创建栈类，采用数组描述；计算数学表达式的值。 输入数学表达式，输出表达式的计算结果。数学表达式由单个数字和运算符“+”、“-”、“\*”、“/”、“(”、“) ”构成，例如 2+3\*(4+5)–6/4。  **输入输出格式**：  **输入**：  第一行一个整数n(1<=n<=100)，代表表达式的个数。  接下来n行，每行一个表达式，保证表达式内的数字为单个整数，表达式内各运算符和数字间没有空格，且表达式的长度不超过2000。  **输出**：  每行表达式输出一个浮点数，要求保留两位小数，保证输入表达式合法。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）   将中缀表达式转后缀表达式：使用符号栈，遍历中缀表达式，遇到优先级比栈顶低的入栈，反之出栈即可。  后缀表达式的计算，使用双精度栈结构，遍历后缀表达式，遇到数压入栈中，反之弹出数据进行计算。   1. 测试结果（测试输入，测试输出）  输入 3  1+6/1\*7+2\*1\*4+9/1+2\*0\*9+9+7/(9\*5)-1\*6-0\*8-7-9\*2+6-(0-5-2\*8-7-9\*5\*(6-5\*5\*2\*6-2-7-5+6\*7+6\*9-1\*0\*0+3\*0+2/1-6/6+5))  0-4-1/6\*(1-(6/7)-4+6+2+6\*1)-1\*7+2-8\*2+0-(4+6-6\*1+(3-8\*6/4-6-5)\*6/4/8+7-1\*4/9\*5)-0/6+1-0-2+7-2+6\*4-3\*6+2/8+6+1\*6\*2  5-3\*9+5/1\*5-9+1\*8-6-8-4\*1+5-2+9/3\*2-2/5/(2-6)\*2/7-9\*0-2+4/6\*6\*7\*8-8-8\*6+8\*9\*(3+0\*1/5/2\*7\*8+0-8\*8-5+8/5\*2-0)  **输出**  -9197.84  -3.47  -4362.57   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   结果正确，输出指定精度的浮点数应使用setprecision   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   #include "bits/stdc++.h"  template <class T>  class stack  {  int stackTop;  int arrayLength;  T \*stackList;  public:  stack();  stack(int initialCapacity);  ~stack() { delete[] stackList; }  bool empty() { return stackTop == -1; }  int size() const { return stackTop + 1; }  T top() const  {  if (stackTop == -1)  exit(-1);  return stackList[stackTop];  }  void pop()  {  if (stackTop == -1)  exit(-1);  stackList[stackTop--].~T();  }  void push(const T &theElement);  };  template <class T>  stack<T>::stack()  {  arrayLength = 100000;  stackList = new T[arrayLength];  stackTop = -1;  }  template <class T>  stack<T>::stack(int initialCapacity)  {  arrayLength = initialCapacity;  stackList = new T[arrayLength];  stackTop = -1;  }  template <class T>  void stack<T>::push(const T &theElement)  {  if (stackTop == arrayLength - 1)  {  T \*temp = new T[arrayLength \* 2];  std::copy(stackList, stackList + arrayLength, temp);  delete[] stackList;  stackList = temp;  arrayLength \*= 2;  }  stackList[++stackTop] = theElement;  }  bool IsPop(char a, char b)  {  if (b == '(')  return false;  if (a == '\*' || a == '/')  {  if (b == '+' || b == '-')  return false;  else  return true;  }  return true;  }  std::string infixtosuffix(std::string src)  {  std::string ans = "";  stack<char> symbol;  size\_t i = 0;  while (i < src.length())  {  if (isdigit(src[i]))  {  ans += src[i];  if (!isdigit(src[i + 1]))  ans += " ";  ++i;  continue;  }  switch (src[i])  {  case ' ':  ++i;  break;  case '(':  symbol.push(src[i]);  ++i;  break;  case ')':  while (symbol.top() != '(')  {  ans += symbol.top();  ans += ' ';  symbol.pop();  }  symbol.pop();  ++i;  break;  case '+':  case '-':  case '\*':  case '/':  if (symbol.empty())  {  symbol.push(src[i]);  ++i;  }  else  {  if (IsPop(src[i], symbol.top()))  {  ans += symbol.top();  ans += ' ';  symbol.pop();  break;  }  else  {  symbol.push(src[i]);  ++i;  break;  }  }  break;  default:  break;  }  }  while (!symbol.empty())  {  ans += symbol.top();  ans += " ";  symbol.pop();  }  return ans;  }  double calculate(std::string suf)  {  stack<double> num;  size\_t i = 0;  double ans = 0;  int temp = 0;  while (i < suf.length())  {  if (suf[i] == ' ')  {  ++i;  continue;  }  else if (isdigit(suf[i]))  {  if (isdigit(suf[i + 1]))  {  temp += suf[i] - '0';  temp \*= 10;  }  else  {  temp += suf[i] - '0';  num.push(temp);  temp = 0;  }  ++i;  continue;  }  else  {  if (num.size() < 2)  exit(-1);  double a = num.top();  num.pop();  double b = num.top();  num.pop();  switch (suf[i])  {  case '+':  num.push(a + b);  break;  case '-':  num.push(b - a);  break;  case '\*':  num.push(a \* b);  break;  case '/':  num.push(b / a);  break;  default:  break;  }  ++i;  }  }  ans = num.top();  return ans;  }  int main()  {  int n = 0;  std::cin >> n;  std::cout.setf(std::ios::fixed, std::ios::floatfield);  while (n--)  {  std::string s;  std::cin >> s;  std::cout << std::setprecision(2) << calculate(infixtosuffix(s)) << std::endl;  }  } | | | |